Incremental Unsupervised Domain-Adversarial Training of Neural Networks

Incremental Unsupervised Domain-Adversarial Training of Neural Networks

Antonio-Javier Gallego[a], Jorge Calvo-Zaragoza[a], Robert B. Fisher[b]

[a]*Department of Software and Computing Systems, University of Alicante, 03690 Spain.*
[b]*School of Informatics, University of Edinburgh, Edinburgh, EH8 9AB, UK.*

Abstract

In the context of supervised statistical learning, it is typically assumed that the training set comes from the same distribution that draws the test samples. When this is not the case, the behavior of the learned model is unpredictable and becomes dependent upon the degree of similarity between the distribution of the training set and the distribution of the test set. One of the research topics that investigates this scenario is referred to as Domain Adaptation (DA). Deep neural networks brought dramatic advances in pattern recognition and that is why there have been many attempts to provide good domain adaptation algorithms for these models. Here we take a different avenue and approach the problem from an incremental point of view, where the model is adapted to the new domain iteratively. We make use of an existing unsupervised domain-adaptation algorithm to identify the target samples on which there is greater confidence about their true label. The output of the model is analyzed in different ways to determine the candidate samples. The selected samples are then added to the source training set by self-labeling, and the process is repeated until all target samples are labeled. This approach implements a form of adversarial training in which, by moving the self-labeled samples from the target to the source set, the DA algorithm is forced to look for new features after each iteration. Our results report a clear improvement with respect to the non-incremental case in several datasets, also outperforming other state-of-the-art domain adaptation algorithms.

Keywords: Domain Adaptation, Unsupervised learning, Neural Networks, Convolutional Neural Networks, Incremental labeling, Self-labeling

1. Introduction

Supervised learning is the most considered approach for dealing with classification tasks. This paradigm is based on a *sufficiently representative* training set to learn a classification model. This level of representativeness is usually defined by two criteria: on the one hand, the training samples

*Corresponding author: Tel.: +349-65-903772; Fax: +349-65-909326
Email addresses: jgallego@dlsi.ua.es (Antonio-Javier Gallego), jcalvo@dlsi.ua.es (Jorge Calvo-Zaragoza), rbf@inf.ed.ac.uk (Robert B. Fisher)

must be varied, which allows the algorithm to generalize instead of memorizing; on the other hand, the application of the trained model is assumed to be carried out on samples that come from the same distribution as those of the training set [1].

Building a training set fulfilling these conditions is not always straightforward. Although obtaining samples might be easy, assigning their correct labels is costly. This is why there are efforts to alleviate the aforementioned requirements. However, while the conflict between memorization and generalization has been well studied, and there exist established mechanisms to deal with it such as regularization or data augmentation [2], learning a model that is able to correctly classify samples from a different target distribution remains open to further research. This problem is generally called *transfer learning* (TL) [3], and when the classification labels do not vary in the target distribution it is usually referred to as *domain adaptation* (DA) [4].

Within the context of supervised learning, deep learning represents an important breakthrough [5]. This term refers to the latest generation of artificial neural networks, for which novel mechanisms have been developed that allow training deeper networks, i.e., with many layers. These deep neural networks are the state of the art in many classification tasks, and have managed to break the existing glass ceiling in many traditionally complex tasks. In turn, deep learning often requires a large amount of training data, which makes the study of DA even more interesting.

As we will review in the next section, there are several approaches to DA, both general strategies and using deep neural networks. In this work we take a different avenue and study an incremental approach. We propose to use an existing DA algorithm to classify those samples of the target domain for which the model is confident. Assuming the assigned pseudo-labels as ground truth, the model is retrained. This idea, also known as self-labeling [6], becomes actually relevant—in addition to simply annotating unlabeled data—since this added knowledge allows the neural network to refine its behavior to correctly adapt to the harder samples of the target set. This incremental process is repeated until the entire target set is completely annotated. We will show that this incremental approach achieves noticeable improvements with respect to both the underlying DA algorithm and the self-labeling procedure. In addition, our proposal is quite competitive on different benchmarks compared to other state-of-the-art DA algorithms.

The rest of the paper is structured as follows: we outline in Section 2 the existing literature about DA, with special emphasis on that based on deep neural networks; we present in Section 3 the proposed incremental methodology, as well as the underlying DA model that we consider in this work; we describe our experimental setting in Section 4, while the results are reported in Section 5; finally, the work is concluded in Section 6.

2. Background

Since the beginning of machine learning research, there exists the idea of exploiting a model beyond its use over unknown samples of the source distribution. In the literature we can find two main topics that pursue this objective: the aforementioned TL and DA strategies.

In TL, some knowledge of the model is used to solve a different classification task. For example, a pre-trained neural network can be used as initialization [7, 8] or its feature extraction process can be considered as the basis of another classification model [9]. As a special case of TL, the DA challenge typically assumes that the classification task of the target distribution is the same (i.e., the set of labels is equal). In this work we focus on the latter case.

In a DA scenario, we can also distinguish between semi-supervised and unsupervised approaches. While semi-supervised DA considers that some labeled samples of the target distribution are available [10, 11, 12], unsupervised DA works with just unlabeled samples [13]. We will revisit in this section unsupervised DA techniques, as it is the case of the proposed approach.

Unsupervised DA is still considered an open problem from both theoretical and practical perspectives [14]. In some way, unsupervised DA can be seen as a sub-case of the semi-supervised learning problem because it also entails learning from both labeled (source domain) and unlabeled (target domain) samples. In this sense, classical ideas for semi-supervised learning can be considered for unsupervised DA. The most representative example is the so-called *self-training* or *self-labeling* approach, where a supervised model is trained from the labeled data and then used to automatically assign a category, often referred to as *pseudo-label*, to each unlabeled sample. In this way, the formerly unlabeled set can be used to train or retrain a model in a supervised fashion, assuming these pseudo-labels as ground-truth. In this context, there are several works that studied this approach for the case of unsupervised DA [15, 16, 17].

A different line of research for unsupervised DA is based on the idea of learning a feature representation that becomes invariant to the domain [18]. A good example is the *Domain Adaptation Neural Network* (DANN) proposed by Ganin et al. [19], which simultaneously learns domain-invariant features from both source and target data and discriminative features from the source domain. Following this line of research, many approaches have been proposed more recently: *Virtual Adversarial Domain Adaptation* (VADA) proposed by Shu et al. [20] added a penalty term to the loss function to penalize class boundaries that cross high-density feature regions. The *Deep Reconstruction-Classification Networks* (DRCN) [21] consists of a neural network that forces a common representation of both the source and target domains by sample reconstruction, while learning the classification task from the source samples. The *Conditional domain adversarial networks* [22] conditions the adversarial learning to discriminative information by two means: multilinear conditioning, that captures the cross-covariance between representations and predictions, and entropy conditioning, that guarantees the transferability by controlling the uncertainty of the predictions.

The *Domain Separation Networks* (DSN) proposed by Bousmalis et al. [23] are trained to map input representations onto both a domain-specific subspace and a domain-independent subspace, in order to improve the way that the domain-invariant features are learned. Haeusser et al. [24] proposed *Associative Domain Adaptation* (ADA), which is another domain-invariant feature learning approach that reinforces associations between source and target representations in an embedding space with neural networks. The *Adversarial Discriminative Domain Adaptation* (ADDA) strategy [25] follows the idea of Generative Adversarial Networks, along with discriminative modeling and untied weight sharing to learn domain-invariant features, while keeping a useful representation for the discriminative task. *Drop to Adapt* (DTA) [26] makes use of adversarial dropout to enforce discriminative domain-invariant features. Damodaran et al. [27] proposed the *Deep Joint Distribution Optimal Transport* (DeepJDOT) approach, which learns both the classifier and aligned data representations between the source and target domain following a single neural framework with a loss functions based on the Optimal Transport theory [28].

A different strategy to DA consists in learning how to transform features from one domain to another. Following this idea, the *Subspace Alignment* (SA) method [29] seeks to represent the source and target domains using subspaces modelled by eigenvectors. Then, it solves an optimization problem to align the source subspace with the target one. Also, Sun and Saenko proposed the *Deep Correlation Alignment* (D-CORAL) approach [30], which consists of a neural network that learns a nonlinear transformation to align correlations of layer activations from the source and target distributions.

While the methods outlined above seek new ways to achieve the desired characteristics of a proper DA method, our proposed approach takes a different avenue. Specifically, we build upon the existing DANN approach and we propose novel ways to improve its ability to adapt to the target domain by performing the adaptation incrementally, inspired by the idea of self-labeling. While the DANN network learns domain-invariant features, adding pseudo-labeled target data to the process incrementally causes these features to become increasingly specialized where there is a larger gap between the source and target distributions. We will see later that the combination of DANN and self-labeling achieves a performance that goes beyond the sum of DANN and self-labeling separately, thus confirming an excellent synergy between these two approaches.

3. Methodology

3.1. Preliminaries

Let X be the input space and Y be the output or *label* space. A classification task assumes that there exist a function $f : X \to Y$ that assigns a label to each possible sample of the input space. For supervised learning, the goal is to learn a hypothesis function h that models the unknown function f with the least possible error. We refer to h as label classifier. Quite often, the approach is to

estimate a posterior probability $P(Y|X)$ so that the label classifier follows a *maximum a posteriori* decision such that $h(x) = \arg\max_{y \in Y} P(y \mid x)$. This is the case with neural networks.

In the DA scenario, there exist two distributions over $X \times Y$: D_S and D_T, which are referred to as *source domain* and *target domain*, respectively. We focus on the case of unsupervised domain adaptation, for which DA is only provided with a labeled source set $S = \{(x_i, y_i)\}_{i=1}^{n} \sim (D_S)^n$ and a completely unlabeled target domain $T = \{(x_i)\}_{i=1}^{n'} \sim (D_T)^{n'}$.

The goal of a DA algorithm is to build a label classifier for D_T by using the information provided in both S and T.

3.2. Domain Adaptation Neural Network

Given its importance in the context of our work, we further describe here the operation of DANN, which will be considered as the backbone for our approach.

DANN is based on the *theory of learning from different domains* discussed by [18, 31]. This suggests that the transfer of the knowledge gained from one domain to another must be based on learning features that do not discriminate between the two domains (source and target). For this, DANN learns a classification model from features that do not encode information about the domain of the sample to be classified, thus generalizing the knowledge from a source labeled domain to a target unlabeled domain.

More specifically, the proposed neural architecture includes a *feature extractor* module (G_f) and a *label classifier* (G_y), which together build a standard feed-forward neural network that can be trained to classify an input sample x into one of the possible categories of the output space Y. The last layer of the label classifier G_y uses a "softmax" activation, which models the posterior probability $P(y \mid x)$, $\forall y \in Y$ of a given input $x \in X$.

DANN adds a new *domain classifier* module (G_d) to the neural network, that classifies the domain to which the input sample x belongs. This classifier is built as a binary logistic regressor that models the probability that an input sample x comes from the source distribution ($d_i = 0$ if $x \sim \mathcal{D}_S$) or the target distribution ($d_i = 1$ if $x \sim \mathcal{D}_T$), where d_i denotes a binary variable that indicates the domain of the sample.

The unsupervised adaptation to a target domain is achieved as follows: the domain classifier G_d is connected to the feature extractor G_f (which is shared with the label classifier G_y) through the so-called *gradient reversal layer* (GRL). This layer does nothing at prediction. However, while learning through back-propagation, it multiplies the gradient by a certain negative constant (λ). In other words, the GRL receives the gradient from the subsequent layer and multiplies it by $-\lambda$, therefore changing its sign before passing it to the preceding layers. The idea of this operation is to force G_f to learn generic features that do not allow discriminating the domain. In addition, since this training is carried out simultaneously with the training of G_y (label classifier), the features

must be adequate for discriminating the categories to classify, yet unbiased with respect to the input domain. According to the DA theory, this should cause G_y to be able to correctly classify input samples regardless of their domain, given that the features from G_f are forced to be invariant.

The DANN training simultaneously updates all modules, providing samples for both G_y and G_d. This can be done by using conventional mechanisms such as Stochastic Gradient Descent, from batches that include half of the examples from each domain. During the training process, the learning of G_f pursues a trade-off between appropriate features for the classification (G_y) and inappropriate features for discriminating the domain of the input sample (G_d). The hyper-parameter λ allows tuning this trade-off. The training is performed until the result converges to a saddle point, which can be found as a stationary point in the gradient update defined by the following equation:

$$\theta_f \leftarrow \theta_f - \mu \left(\frac{\partial \mathcal{L}_y}{\partial \theta_f} - \lambda \frac{\partial \mathcal{L}_d}{\partial \theta_f} \right) \tag{1}$$

where θ_f denotes the weights of G_f, μ denotes the learning rate, and $\mathcal{L}_y$ and $\mathcal{L}_d$ represent the loss functions for the label classifier and the domain classifier, respectively.

A graphical overview of the DANN architecture is depicted in Fig. 1.

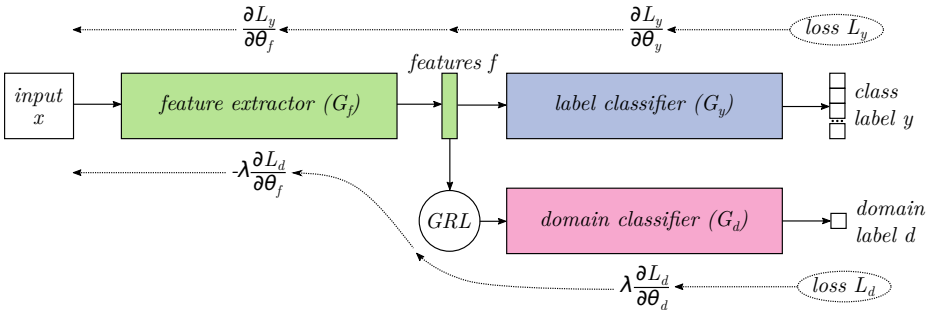


Figure 1: Graphical overview of the DANN architecture, consisting of three blocks: feature extractor (G_f), label classifier (G_y), and domain classifier (G_d). The GRL circle denotes the gradient reversal layer that multiplies the gradient by a negative factor.

3.3. Incremental DANN

Our main contribution within the context of DA is to propose an incremental approach to DANN (iDANN). This strategy is explained below.

Once the DANN model is trained as explained in the previous section, we can use both the feature extractor G_f and the label classifier G_y to predict the category of samples from both the target domain and the source domain ($G_y(G_f(x))$). The "softmax" activation used at the output of this classifier returns the posterior probability that the network considers x to belong to any of the classes of the output space Y.

Our main assumption is that we can use the subset of samples from the target domain for which G_y is more confident, and then add them to the source labeled domain assuming the prediction as

ground truth. These samples are thereafter considered as samples of the source domain completely. Afterwards, we can retrain the DANN network to fine-tune its weights using the new training set. This process is repeated iteratively, moving the labeled samples with greater confidence from the target domain to the source domain after each iteration. We stop when there are no more samples to move from the target domain.

The intuitive idea behind our approach is that by adding target domain information to the source (labeled) domain, the DANN learns new domain-invariant features that better fit the eventual classification task, thereby becoming more accurate for other target domain samples. In each iteration, however, the task increases its complexity because it deals first with the simplest samples to classify (for which the DANN is more confident), leaving those that have more dissimilar features in the unlabeled target set. When the DANN is retrained with labeled samples that include target domain information, the domain classifier G_d needs to be more specific. This forces the feature extraction module G_f to forget the features that differentiate more complex samples from the target domain.

We formalize the process in Algorithm 1, where e and b represents the number of epochs and the batch size considered, respectively, e_{inc} denotes the number of epochs for the incremental stage of the algorithm, r indicates the size of the subset of target domain samples to select in each iteration, and β is a constant that allows us to modify this size after each iteration.

Algorithm 1: Incremental DANN (iDANN)

Input : $S \leftarrow \{(x_i, y_i) \sim \mathcal{D}_S\}$
$T \leftarrow \{(x_i) \sim \mathcal{D}_T\}$
$e, e_{inc}, b, r, \lambda, \beta \leftarrow$ Initial hyper-parameters values

Output: G_f, G_y, G'_f, G'_y

while $T \neq \emptyset$ **do**
  $G_f, G_y \leftarrow$ Fit DANN with $\{S, T, e, b, \lambda\}$
  $\hat{B}_r \leftarrow$ `selection_policy`(G_f, G_y, T, r)
  $S \leftarrow S \cup \hat{B}_r$
  $T \leftarrow T \backslash \hat{B}_r$
  $e \leftarrow e_{inc}$
  $r \leftarrow \beta\, r$
end while
$\hat{T} \leftarrow \{(x_i, y_i) \mid x_i \sim \mathcal{D}_T, y_i = G_y(G_f(x_i))\}$
Fit $G'_y(G'_f(\cdot))$ with $\{\hat{T}, e, b\}$

In this algorithm, the samples of the target domain ($\hat{B}$) are classified using the label classifier G_y, and then it proceeds to select a subset $\hat{B}_r$ of size r to be moved from the target domain to the source domain. For this purpose, two selection criteria are proposed, which are described in the next

section.

Once the iterative stage of the algorithm ends, the label classifier G_y is used to classify the entire original target domain (see line 9 of Algorithm 1). This labeled target set is used to then train a neural network from scratch, which is therefore specialized in classifying target domain samples (more details in Section 3.5).

3.4. Selection policies

Below we describe in detail the two proposed policies to select samples during the iterative stage of Algorithm 1 (`selection_policy`). One policy is directly based on the confidence level that the network provides to the prediction, while the other is based on geometric properties of the learned feature space.

3.4.1. Confidence policy

As mentioned above, the output of the label classifier G_y uses a softmax activation. Let L denote the number of labels. Then, the standard softmax function $\sigma : \mathbb{R}^L \to \mathbb{R}^L$ is defined by Equation 2.

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{L} e^{z_j}} \text{ for } i = 1, \ldots, L \quad \text{and } \mathbf{z} = (z_1, \ldots, z_L) \in \mathbb{R}^L \tag{2}$$

This function normalizes an L-dimensional vector $\mathbf{z}$ of unbounded real values into another L-dimensional vector $\sigma(\mathbf{z})$, for which values range between $[0, 1]$ and add up to 1. This can be interpreted as a posterior probability over the different possible labels [32]. In order to turn these probabilities into the predicted class label, we simply take the argmax-index position of this output vector, following a *Maximum a Posteriori* probability criterion.

Taking advantage of this interpretation, the first policy for selecting samples to move from the target domain to the source is based on the probability provided by the label classifier G_y, which can be seen as a measure of confidence in such classification.

With this criterion, we will keep the maximum predicted probability value for each sample of the target set among the possible labels. Then, we will order all samples based on this value—from highest to lowest—in order to select the first r samples to build the subset $\hat{B}_r$.

Algorithm 2 presents the algorithmic description of this process, where G_y^p refers to the probabilistic output of the label classifier after the softmax activation, before applying argmax to select a label. The function *sortr* is used to sort the set in decreasing order.

Figure 2 shows an example of a set of probabilities obtained after predicting the target samples with DANN. The figure on the left shows the maximum probability values obtained for the classification of each sample—without sorting—while in the figure on the right the sorted set is shown, where the threshold r has been highlighted.

Algorithm 2: Confidence policy

Input : $T \leftarrow \{(x_i) \sim \mathcal{D}_T\}$

$G_f, G_y \leftarrow$ Feature extractor and label classifier

$r \leftarrow$ Size of the selected samples subset

Output: $\hat{B}_r$

```
B̂ ← G_y^p(G_f(T))
B̂_r ← {∅}
foreach (x_i, y_i) ∈ sortr(B̂) do
    B̂_r ← B̂_r ∪ (x_i, y_i)
    if |B̂_r| = r then
        break
    end if
end foreach
```

3.4.2. kNN policy

As in the previous case, once the network has been trained, we use the label classifier G_y to predict the labels of the whole target domain and then we sort them based on the confidence given by the network. However, in this case, instead of directly selecting a subset of samples according to this confidence, we will also evaluate the geometric properties of the feature space. This is performed following the k-nearest neighbor rule.

We first obtain the feature set F_S from the source set S (using $G_f(S)$). We then proceed to iterate the target set samples sorted by their level of confidence. Given a target sample, if the label of the k-nearest samples of the source domain matches the label assigned by the label classifier G_y, then we will select the prototype. Otherwise, we will discard it. Therefore, samples are selected based on both the confidence provided by the DANN in their label and the extent they match the distribution of the source domain.

Algorithm 3 describes this process. The $kNN(q, F_S, k)$ function receives as parameters the query sample q, the set F_S and the value k to be used, and yields the predicted label l and the number of samples m within its k-nearest neighbors from S that have the same label.

The idea of this policy is to select the samples of the target domain whose features are within the cluster of the source domain for the same class. An illustrative example of this condition is shown in Fig. 3 with $k = 5$. The example shows two labels of the source domain as green circles and blue squares. The red stars denote the target domain examples that are being evaluated to determine if they are selected. For instance, the star on the left would be selected if, and only if, the network classified it as a green circle, since its 5-nearest neighbors are green circles. Similarly, the star on the right would be selected if, and only if, the network classified it as a blue square. However, the

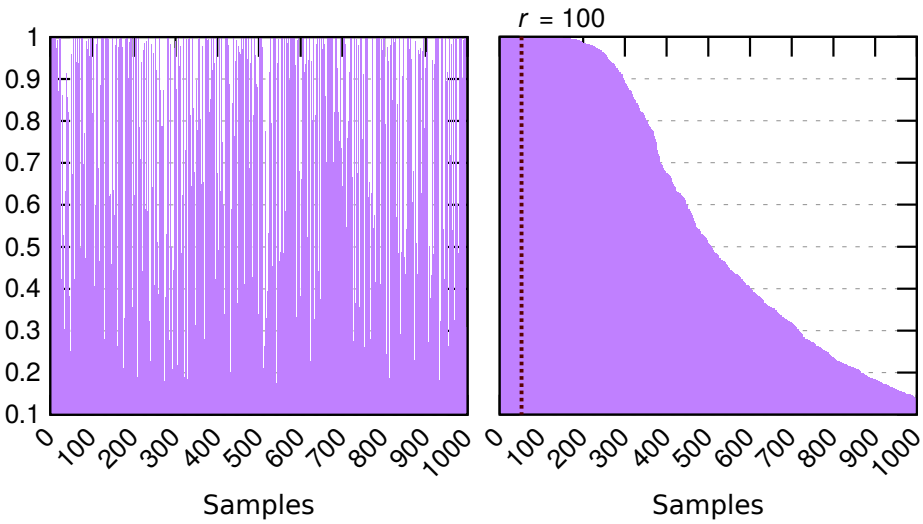


Figure 2: Example of probabilities obtained with DANN. Left: maximum probability of each sample. Right: ordered set of maximum probabilities, where the threshold r has been highlighted.

central star would always be discarded because its 5-neighbors belong to two different classes.

If we increase k, the red star of the left would still be selected (if labeled as green circle) because it is located in the middle of the cluster. However, the red start of the right is closer to label boundaries, and so it would eventually be discarded.

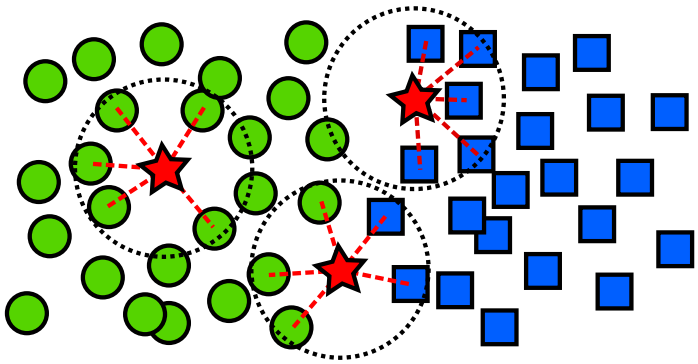


Figure 3: Example of sample selection using the kNN policy with $k = 5$. Green circles and blue squares represent samples of two different classes from the source domain. Red stars represent the samples of the target domain that are evaluated to determine whether they are chosen.

3.5. Training a classifier with the new labeled target set

As described in Algorithm 1, once the iterative stage of the iDANN algorithm is completed, we use the label classifier G_y to annotate the entire original target set T from scratch. Then, another neural network is trained by conventional means considering the same neural architecture of $G_y(G_f(\cdot))$, which we refer to as $G'_y(G'_f(\cdot))$. Note that we assume the same topology but this is not strictly necessary. This new network can be trained from scratch or starting from the weights

Algorithm 3: kNN policy

Input : $S \leftarrow \{(x_i, y_i) \sim \mathcal{D}_S\}$
$T \leftarrow \{(x_i) \sim \mathcal{D}_T\}$
$G_f, G_y \leftarrow$ Feature extractor and label classifier
$r \leftarrow$ Size of the selected samples subset
$k \leftarrow$ Number of neighbors to consider

Output: $\hat{B}_r$

$F_S \leftarrow G_f(S)$
$\hat{B} \leftarrow G_y^p(G_f(T))$
$\hat{B}_r \leftarrow \{\emptyset\}$
foreach $(x_i, y_i) \in sortr(\hat{B})$ **do**
  $f_T^{(i)} \leftarrow G_f(x_i)$
  $l, m \leftarrow kNN(f_T^{(i)}, F_S, k)$
  if $y_i = l$ ***and*** $m = k$ **then**
    $\hat{B}_r \leftarrow \hat{B}_r \cup (x_i, y_i)$
    if $|\hat{B}_r| = r$ **then**
      break
    end if
  end if
end foreach

of the incremental learning process. Either way, the objective is to eventually get a model that is directly specialized in the classification of the target domain.

However, we assume that some part of the iterative annotation of the target set will contain noise at the label level. To mitigate the possible efects of this noise, we consider *label smoothing* [33]. This is an efficient and theoretically-grounded strategy for dealing with label noise, which also makes the model less prone to overfitting.

Compared to the classical one-hot output representation, denoted by $\mathbf{y}$, label smoothing changes the construction of the true probability to

$$\mathbf{y}_i' = (1 - \epsilon)\mathbf{y}_i + \frac{\epsilon}{L}, \tag{3}$$

where ϵ is a small constant (or smoothing parameter) and L is the total number of classes. Hence, instead of minimizing cross-entropy with hard targets (0 or 1), it considers soft targets.

3.6. Insights into the learning process

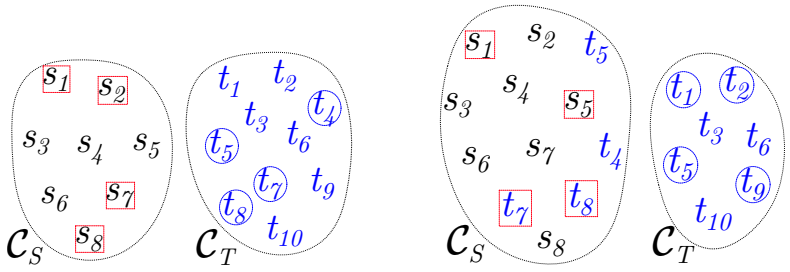
To clarify how the incremental step collaborates with the DA step, we will provide in this section an abstract explanation behind the fundamentals of the proposed learning process, taking into account the discriminative features of both the source and the target domains.

Let us represent the discriminative features of domains S and T as $\mathcal{C}_S = \{s_1, s_2, ..., s_n\}$ and $\mathcal{C}_T = \{t_1, t_2, ..., t_m\}$. The elements of $\mathcal{C}_S$ and $\mathcal{C}_T$ represent specific features that define each domain for pattern recognition purposes. That is, these features can be used to discriminate the classes and/or to identify the domain itself. For instance, in the case of images, an element of these sets might represent different types of borders, colors, color gradients, etc., which we hope the neural network identifies during the learning process.

To successfully apply DA between these two domains, we must assume that there is a subset $\mathcal{I} = \mathcal{C}_S \cap \mathcal{C}_T$ with the features that are common to both—i.e., $\mathcal{I} \neq \varnothing$—which would correspond to the domain-invariant features that the DANN algorithm is supposed to learn for classifying the target samples in an unsupervised way. This also implies that there is a subset $\mathcal{C}_S \setminus \mathcal{C}_T$ with the domain-dependent features of S and another subset $\mathcal{C}_T \setminus \mathcal{C}_S$ with the domain-dependent features of T, which are useful for discriminating between domains.

Figure 4 visually represents a fictional example of a composition of $\mathcal{C}_S$ and $\mathcal{C}_T$ sets in the first two iterations of our algorithm. Within these two sets, we highlight a subset of the features that the domain classifier G_d might use to differentiate between the domains, that is, the domain-dependent features from each domain. The domain-dependent features of S are marked with red boxes, while the domain-dependent features of T are marked with blue circles. These are the features that, through the GRL layer, are forced to be ignored by the feature extractor G_f. This operation makes these domain-dependent features not be used by the label classifier G_y; however, this does not guarantee that the rest of the features that have not been learned (those that are not highlighted in Fig. 4a) belong to the set $\mathcal{I}$ of domain-invariant features, but there might be some domain-dependent features that G_d has not considered because they are not necessary. In other words, there might be features that allow the domains to be easily differentiated—such as the background color—which would be enough at the beginning of the process. In this case, G_d would only need this subset of domain-dependent features, thus ignoring more complex domain-dependent features in the first steps.

In subsequent iterations, $\mathcal{C}_S$ and $\mathcal{C}_T$ vary because some target samples are moved from the target domain to the source domain (Fig. 4b)—after assigning pseudo-labels—and, therefore, the features that are domain-invariant do not necessarily remain the same. In this example, features t_4, t_5, t_7, and t_8 are moved from $\mathcal{C}_T$ to $\mathcal{C}_S$ after the first step. Note that, some of these features might stay in $\mathcal{C}_T$, depending on whether any of the remaining target samples are identified by these features or not. In the following iteration, the DANN needs to look for new domain-dependent features as



(a) 1st iteration (b) 2nd iteration

Figure 4: Figurative example depicting the set of discriminative features in two iterations of the proposed algorithm. The elements of $\mathcal{C}_S$ and $\mathcal{C}_T$ represent specific visual features that can be used to discriminate the classes and/or to identify the domain itself. The subset of features that the algorithm would use to discriminate the domains are highlighted: the domain-dependent features of S are marked with red boxes, while the domain-dependent features of T are marked with blue circles. After the first iteration, the iDANN algorithm moves some selected samples ($\hat{B}_r$) from the target set to the source set and, because of this, some features are moved from $\mathcal{C}_T$ to $\mathcal{C}_S$ as well.

the simplest ones could no longer be used to differentiate between domains. This also produces a positive side effect: given that there are (pseudo-)labeled samples that contain common features with the target set (for instance, t_5 in our example), these help to correctly classify unlabeled target samples. This will be seen experimentally in Section 5.2.

The process is repeated iteratively, moving samples from the target to the source set, thus also moving features from $\mathcal{C}_T$ to $\mathcal{C}_S$. After each iteration, the iDANN algorithm will move the domain-dependent features from the target domain to the source domain, leaving only the common features (those that are domain-invariant) or the domain-dependent features that are more complex to learn. As aforementioned, in the first iterations, the domain discriminator G_d only needs the simplest features to differentiate between domains, but after these features are transferred to the source, G_d will search for others more complex or more specific to look at. Eventually, this causes the GRL layer to make G_f forget the most complex domain-dependent features. Finally, in the last steps, the iterative algorithm assumes that the network prediction is correct for the most complex features and ends up adding them to the source as well.

4. Experimental setup

4.1. Datasets

The proposed approach will be evaluated with two different classification tasks, that are common in the DA literature. The first one is that of digit classification, for which we consider the following datasets:

- MNIST [34]: this collection contains 28×28 images representing isolated handwritten digits.

- MNIST-M [19]: this dataset was synthetically generated by merging MNIST samples with random color patches from BSDS500 [35].

- Street View House Numbers (SVHN) [36]: it consists of images obtained from house numbers from Google Street View. It represents a real-world challenge of digit recognition in natural scenes, for which several digits might appear in the same image and only the central one must be classified.

- Synthetic Numbers [19]: images of digits generated using Windows^{TM} fonts, with varying position, orientation, color and resolution.

In addition, we also evaluate our approach for traffic sign classification with the following datasets:

- German Traffic Sign Recognition Benchmark (GTSRB) [37]: this dataset contains images of traffic signs obtained from the real world in different sizes, positions, and lighting conditions, as well as including occlusions.

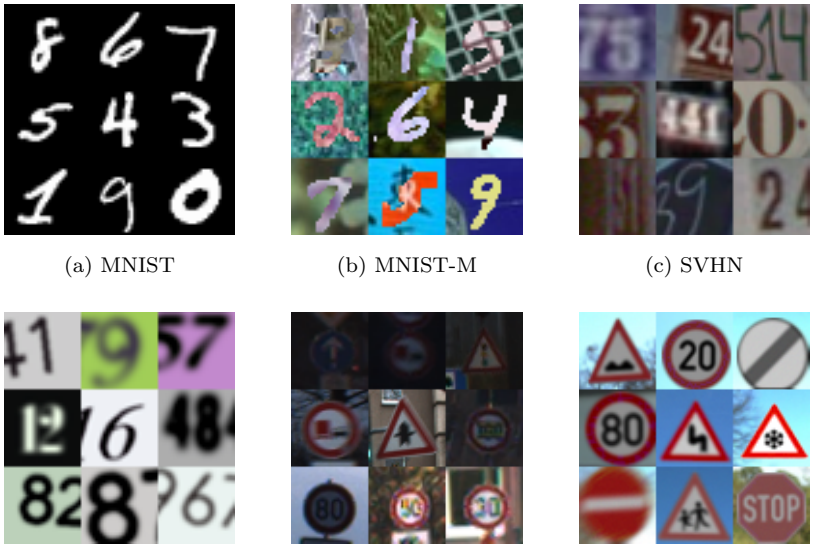
- Synthetic Signs [38]: this dataset was synthetically generated by taking common street signs from Wikipedia and applying several transformations. It tries to simulate images from GTSRB although there are significant differences between them.

Table 1 summarizes the information of our evaluation corpora, including the domain to which they belong, the number of labels, the image resolution, and the number of samples. Figure 5 shows some random examples from each of these datasets.

Table 1: Description of the datasets used in the experimentation.

Set	**# labels**	**Domain**	**Resolution (px)**	**# samples**
Digits	10	MNIST	28×28	65,000
		MNIST-M	28×28	65,000
		SVHN	32×32	99,289
		Syn. Numbers	32×32	488,953
Traffic signs	43	GTSRB	[25×25, 225×243]	51,839
		Syn. Signs	40×40	100,000

The images of each classification task were rescaled to the same size: the digits to 28×28 pixels, and the traffic signs to 40×40 pixels. Concerning the pre-processing of the input data, the RGB channels of each image were independently normalized within the range $[0, 1]$. The train and test partitions were those proposed by the authors of each dataset, in order to ensure a fair comparison with the results obtained in the literature.



(a) MNIST (b) MNIST-M (c) SVHN

(d) Syn. Numbers (e) GTSRB (f) Syn. Signs

Figure 5: Random examples from the datasets used in experimentation.

4.2. Neural architectures

To evaluate the proposed methodology, the same three Convolutional Neural Network (CNN) architectures considered in the original DANN paper have been tested. Table 2 reports a summary of these architectures.

As the authors pointed out, these topologies are not necessarily optimal and better adaptation performance might be attained if they were tweaked. However, we chose to keep the same configuration to make a fairer comparison.

As the activation function, a Rectifier Linear Unit (ReLU) was used for each convolution layer and fully-connected layer, except for the output layers. L neurons with softmax activation were used as output of the label classifier. For the output of the domain classifier, a single neuron with a logistic (sigmoid) activation function was used to discriminate between two possible categories (source domain or target domain).

Model 1 was used for all the experiments with digit datasets, except those using SVHN. This topology is inspired by the classical LeNet-5 architecture [34]. Model 2 was used to evaluate the experiments with digits that include SVHN. This architecture is inspired by [39]. Finally, Model 3 was used for the experiments with traffic sings. In this case, the single-CNN baseline obtained from [40] was used.

4.3. Training stage

To ensure a fair comparison with the original DANN algorithm, we set the same training configuration: Stochastic Gradient Descent with a learning rate of 0.01, decay of 10^{-6}, and momentum of 0.9, as well as the same number of epochs (300).

Table 2: CNN network configurations considered. Notation: $Conv(f, w, h)$ stands for a layer with f convolution operators with a kernel of size $w \times h$ pixels, $MaxPool(w, h)$ stands for the max-pooling operator of dimensions $w \times h$ pixels—with 2×2 in all cases—and $FC(n)$ represents a fully-connected layer of n neurons. In the output layer of the label classifier a fully-connected layer of L neurons with softmax activation is added, where L denotes the number of categories of the dataset at issue.

	Feature extractor				
Model	**Layer 1**	**Layer 2**	**Layer 3**	**Label classifier**	**Domain classifier**
1	Conv(32, 5, 5) MaxPool(2, 2)	Conv(48, 5, 5) MaxPool(2, 2)		FC(100) FC(100) FC(L)	FC(100) FC(1)
2	Conv(64, 5, 5) MaxPool(3, 3)	Conv(64, 5, 5) MaxPool(3, 3)	Conv(128, 5, 5)	FC(3072) FC(2048) FC(L)	FC(1024) FC(1024) FC(1)
3	Conv(96, 5, 5) MaxPool(2, 2)	Conv(144, 3, 3) MaxPool(2, 2)	Conv(256, 5, 5) MaxPool(2, 2)	FC(512) FC(L)	FC(1024) FC(1024) FC(1)

To determine e_{inc}, r, and β, a detailed analysis is provided in the experimentation section, eventually setting them to 25, 5%, and 1.5, respectively. Different values for both the batch size b and λ are evaluated, as well.

5. Results

In this section we evaluate the proposed method and two selection policies using the datasets, topologies, and settings described in Section 4. We first study the different hyper-parameterization, as well as the two prototype selection policies proposed. Next we show the performance results obtained over the datasets and, finally, we compare with other state-of-the-art methods.

5.1. Hyper-parameters evaluation

In this section, we start by analyzing the influence of the batch size and the value of λ on the performance of the method, as these hyper-parameters are those that affect the training stage the most. For this, we consider the batch sizes of $\{16, 32, 64, 128, 256, 512\}$ and λ of $\{10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}\}$. This means that each result comes from a total of 336 experiments (14 combinations of dataset pairs $\times$ 6 batch sizes $\times$ 4 values of λ). The rest of hyper-parameters are set as indicated in Section 4.3, that is: $e = 300$ (as in the original DANN paper), $e_{inc} = 25$, and $r = 5\%$, which were empirically determined to favor stable training and obtain good results. In addition, we evaluate the results using only the prototype selection policy based on network's confidence, as next section will be devoted to comparing the two proposed policies with the best hyper-parameters found.

As we are dealing with an unsupervised method, we mainly focus on analyzing the trend when modifying these parameters. Table 3 shows the results of this experiment, where each figure represents the average of the 14 possible combinations of source and target domain of the datasets considered and all the iterations performed by the iDANN algorithm.

The first thing to remark is that some of the hyper-parameter combinations evaluated in these experiments do not converge (batch = 32/64, $\lambda = 10^{-1}$ for traffic signs). This could be detected automatically, since the accuracy is abruptly reduced to a value approximately equal to a random guess, for both the training set and evaluation set and for both the source and the target domain. However, these results have been kept in order to observe the general trend of the method and how these parameters affect it.

It can also be observed that the best performance is achieved with $\lambda = 10^{-2}$ in the two types of corpora, while a batch size of 64 and 32 are better for the digits and traffic signs, respectively. On average, better results are reported with low λ values and batch sizes between 32 and 256. When λ is greater (e.g., 10^{-1}), the training becomes highly unstable, especially if combined with small batch sizes.

Table 3: Influence of hyper-parameter setting on the performance (accuracy, in %) of the iDANN algorithm. Figures report classification accuracy over the target set, averaging with the respective datasets and iterations of the algorithm.

	Numbers			
	λ			
Batch	$\mathbf{10^{-4}}$	$\mathbf{10^{-3}}$	$\mathbf{10^{-2}}$	$\mathbf{10^{-1}}$
16	58.74	56.21	58.65	47.54
32	66.13	65.82	61.67	49.78
64	65.26	66.41	**66.82**	62.54
128	64.23	66.04	66.79	52.89
256	64.55	63.94	64.24	59.36
512	62.55	62.61	62.75	50.67
	Traffic signs			
	λ			
Batch	$\mathbf{10^{-4}}$	$\mathbf{10^{-3}}$	$\mathbf{10^{-2}}$	$\mathbf{10^{-1}}$
16	89.67	88.65	90.08	48.16
32	93.58	93.63	**94.50**	24.02
64	91.27	91.16	91.67	31.41
128	88.56	89.36	89.60	66.78
256	87.34	87.73	88.67	91.39
512	84.34	84.45	84.09	84.60

Next, we analyze the influence of these parameters with respect to the iteration of the iDANN algorithm. Table 4 shows the average result obtained by grouping all combinations of datasets (digits and traffic signs) and hyper-parameters considered. As in the previous analysis, better results are also observed for low λ values and batch sizes between 32 and 256 (see column 'Avg.'). In this case, it can also be seen that low λ values are more appropriate in the first iterations, whereas greater λ values are more appropriate in the last iterations. It might happen that a more stable way of proceeding (low λ) is preferred in the first iterations, even at the cost of being less aggressive in the domain adaptation. Therefore, we propose to start with a low λ and increase its value gradually ($+10^{-4}$ after each epoch).

Additionally, it is observed that each iteration of the algorithm leads to a better result than the previous one (except for $\lambda \geq 10^{-1}$), yielding the higher leap in the first iterations and reducing this difference towards the last iterations. Including all cases, the results improve by 5.19% between the first and the last iteration, on average. If we ignore those settings that do not converge, the average improvement obtained increases to 10.29%.

Other hyper-parameters of the proposed method that might affect the result obtained are e_{inc}, β, and r. This is why we perform below a sensitivity analysis of each of these variables independently.

The value e_{inc} adjusts the number of training epochs during the iterative stages. The following set of values $e_{inc} = \{1, 5, 10, 15, 25, 50, 100, 300\}$ is analyzed. Figure 6a shows the average result obtained for the datasets of numbers and traffic signs using the best configuration obtained previously. As can be observed, the greatest improvement is given at the beginning (up to 15 epochs), then the result stabilizes. Eventually, we decided to set $e_{inc} = 25$ with the intention of not increasing the training time unnecessarily.

β and r configure the training schedule during the iterative stages, since we can control the number of iterations or the number of samples included in the source set in each iteration as we modify these variables. To evaluate these hyper-parameters, the following sets are considered: $\beta = \{1, 1.1, 1.2, 1.4, 1.5, 1.7, 2, 4, 6\}$ and $r = \{0.1, 0.2, 0.5, 1, 3, 5, 7, 10, 20, 50\}$. Figure 6b shows the results obtained by varying the parameter β for the datasets considered. In addition, the number of iterations is also shown (represented on the right vertical axis). As observed, the accuracy remains stable up to a value of 1.7, decreasing slightly when set to 2, and then falling abruptly. Furthermore, it is observed that the number of iterations is very high for low values, so eventually β is set to 1.5. Figure 6c reports the same experiment when varying parameter r. A similar phenomenon is observed: a stable accuracy is obtained for small values, up to $r = 7$, slightly worsening for $r = 10$, and decreasing for higher values. It can also be seen how the number of iterations is very high for small values of r. We decided to set $r = 5$ because it gets a similar result than lower values but is more efficient (fewer number of iterations).

Table 4: Influence of hyper-parameter setting on the performance of the iDANN algorithm with respect to number of iterations. Figures report classification accuracy over the target set, averaging with the respective datasets.

		Iterations									
$\boldsymbol{\lambda}$	**Batch**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**Avg.**
$\mathbf{10^{-4}}$	16	58.46	59.71	61.46	62.81	63.91	64.86	65.39	65.78	66.04	63.16
	32	65.20	67.85	69.37	69.91	70.73	71.14	71.89	72.11	**72.24**	**70.05**
	64	63.88	67.11	68.33	68.75	69.35	70.30	70.71	71.13	71.22	68.97
	128	62.67	65.52	67.09	67.68	68.19	68.84	69.46	69.88	70.06	67.71
	256	62.82	65.20	66.92	67.83	68.65	68.84	69.57	70.09	70.34	67.81
	512	61.51	63.28	64.32	65.45	65.97	66.93	67.60	67.87	68.03	65.66
$\mathbf{10^{-3}}$	16	56.65	57.70	59.65	60.72	61.43	62.25	62.75	63.15	63.31	60.85
	32	63.95	67.42	68.68	69.93	70.59	71.26	71.79	72.19	72.33	69.79
	64	64.66	67.39	68.78	69.89	70.41	71.32	72.06	72.44	**72.57**	**69.95**
	128	63.75	66.59	68.61	69.07	69.93	70.82	71.44	72.00	72.14	69.37
	256	62.38	65.08	66.67	67.44	67.69	68.52	69.10	69.49	69.71	67.34
	512	61.36	63.46	64.72	65.48	66.25	66.96	67.42	67.84	68.08	65.73
$\mathbf{10^{-2}}$	16	56.73	61.57	63.37	65.13	65.81	66.31	62.94	63.13	63.26	63.14
	32	62.07	64.75	66.01	66.68	67.46	67.78	68.23	68.47	68.80	66.69
	64	64.78	67.77	69.44	70.20	71.21	71.92	72.35	72.74	**72.95**	**70.37**
	128	64.49	67.27	69.02	69.75	70.71	71.58	72.00	72.72	72.87	70.05
	256	63.16	65.55	66.75	67.49	68.32	68.76	69.51	69.81	70.21	67.73
	512	61.61	63.49	64.82	65.57	66.10	66.81	67.52	68.01	68.28	65.80
$\mathbf{10^{-1}}$	16	46.48	50.71	52.12	53.35	53.78	42.41	42.98	43.35	43.52	47.63
	32	50.09	53.07	47.66	42.61	43.12	44.06	44.39	44.91	44.96	46.10
	64	61.64	64.02	64.24	54.01	54.48	55.44	55.96	56.47	56.59	58.09
	128	55.72	57.10	57.16	57.01	52.95	53.15	53.63	53.50	53.69	54.88
	256	60.13	62.26	62.60	63.53	64.25	64.93	65.57	66.04	**66.14**	**63.94**
	512	53.54	54.39	54.63	55.01	55.69	56.04	56.57	56.84	56.95	55.52
Average		60.32	62.84	63.85	63.97	64.46	64.63	65.03	65.42	**65.59**	–

5.2. Model analysis

We now evaluate the effect of the incremental training process on the domain adaptation approach. Figure 7 shows the evolution of the accuracy obtained over the target test set during the

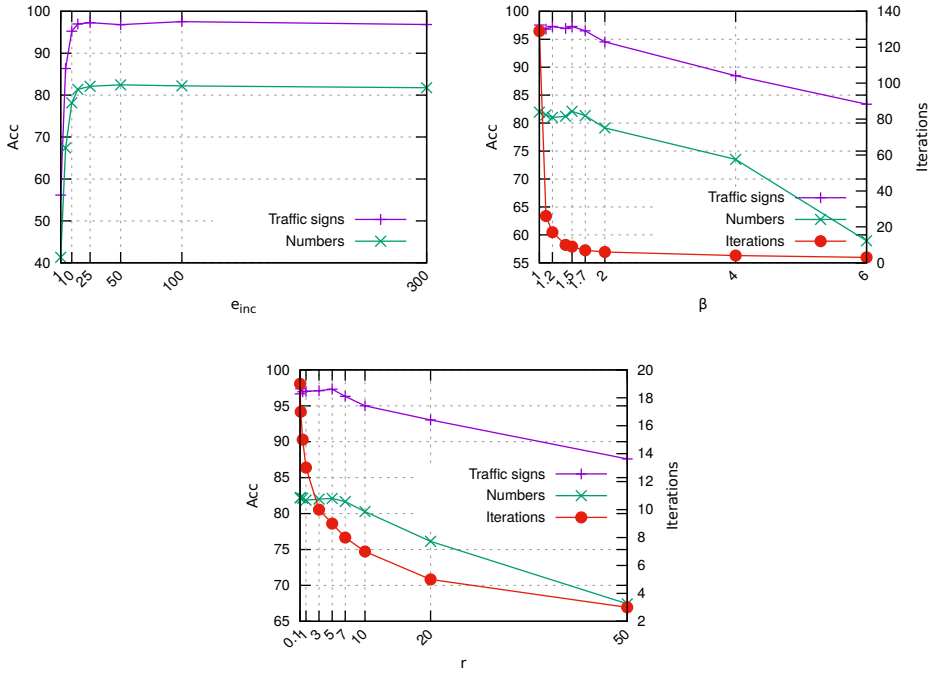


Figure 6: Sensitivity analysis of hyper-parameters (a) e_{inc}, (b) β and (c) r over the digits and traffic signs datasets. In addition, the number of iterations (represented on the right vertical axis) is also reported when varying for β and r.

training process for the case Syn Numbers → MNIST-M combination of datasets, with a batch size of 64 and $\lambda = 10^{-2}$. The training epochs are represented with the horizontal axis, while the iterations (i.e., when new training samples are added) are highlighted with blue lines and marked above. Iteration 1 does not contain any target domain samples in the training set. It can be observed that in the first iteration (spanning 300 epochs), the accuracy slowly improves until around 150 epochs, after which becomes stable. In the subsequent iterations, the accuracy further improves, especially during iterations 2, 3 and 4. Then, the performance increase is gradually reduced until it is hardly noticeable.

To provide further analysis, we also examine the representation space learned by the network in each of these iterations, using the same combination of datasets and training parameters. We use the t-Distributed Stochastic Neighbor Embedding (t-SNE) [41] projection to visualize the samples according to their representation by the last hidden layer of the label predictor. Figure 8 shows a visualization of the features learned after each of the iterations, where the red color represents the target domain, the blue color represents the source domain, and the green color represents the set $\hat{B}_r$ (selected samples) using the confidence policy. This representation reveals 10 well-defined clusters—the 10 possible classes of the datasets considered for this analysis—around an additional central cluster. This central cluster groups the samples of the target domain (red color)

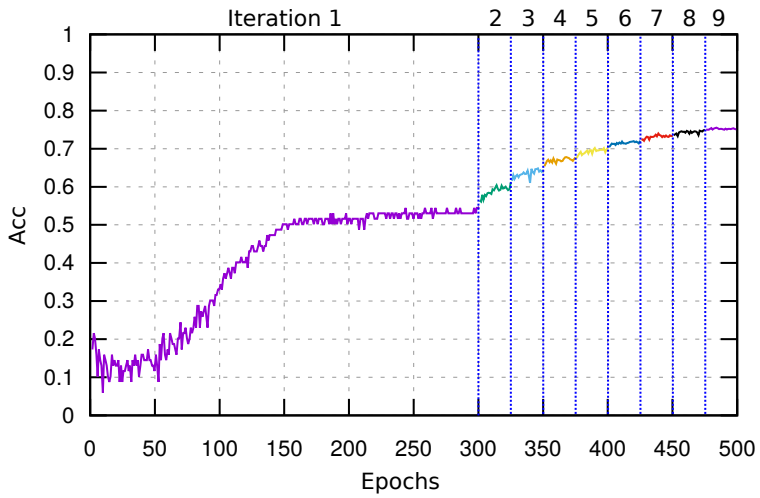


Figure 7: Accuracy curve with respect to training epochs and iterations of the incremental approach.

whose representation does not correspond to any of the existing classes yet. This cluster would therefore correspond to target samples whose representation has not been correctly mapped onto any of the source domain classes. Iteratively, the method is selecting samples (green points) of the target domain and moving them to the source domain. In the first iterations—until the 6th one, approximately—the method selects only samples that are well located in one of the source domain clusters (that is, those samples for which the network is more confident). Due to this process, the size of the central cluster is reduced. It is important to emphasize that this cluster becomes smaller although no samples out of it are selected, which indicates that the network is learning to better map those samples because of the selected samples of previous iterations. Towards the last iterations, the method begins to select the most complex samples that are still in this additional cluster. In Fig. 8(*) (which is the same as the Fig. 8(9) but highlighting each class with a different color), the additional cluster of target samples still appears without being mapped, yet with a very small size. This cluster contains almost all the classification errors, having mapped only some isolated prototypes to the actual class clusters incorrectly.

5.3. kNN policy

We compare in this section the two policies proposed for selecting the set of target prototypes $\hat{B}_r$ to be added to the source domain. To this end, we evaluate whether the label assigned to each of these prototypes is correct. In this case, we make use of the ground-truth of the target domain just for the sake of analysis.

We show in Fig. 9 a dotted line with the performance of the confidence policy, which may serve as a baseline here, and eight results for the kNN policy with varying k values. As in the previous experiments, the reported figures are obtained for all combinations of datasets and hyper-parameters

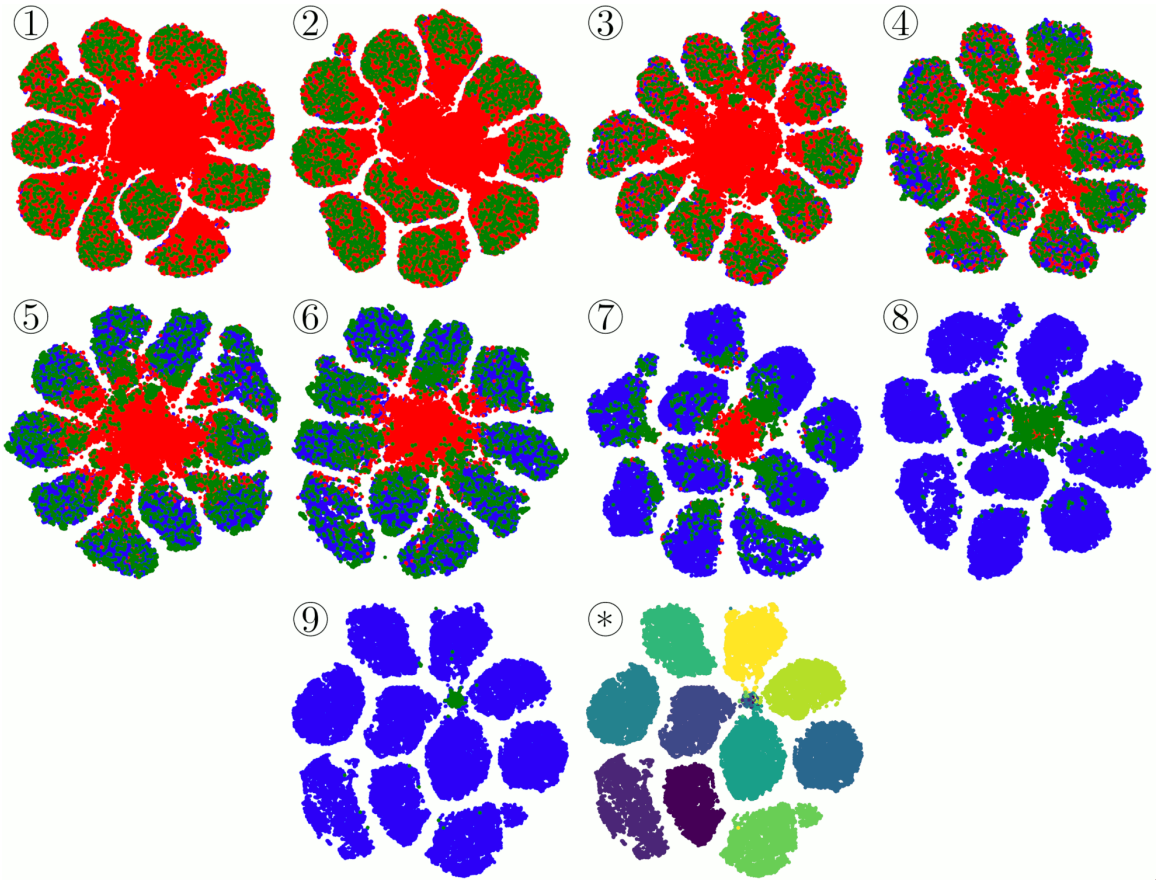


Figure 8: t-SNE representation of the feature space of the neural network (from the last hidden layer) with respect to the iteration of the approach. The red color represents the target domain, the blue color represents the source domain, and the green color represents the set $\hat{B}_r$ that is selected to be added in the next iteration. Last representation (*) depicts each sample according to its actual category by using different colors.

considered.

It is observed that, as the number of iterations of the algorithm increases, the accuracy of the additional labels assigned to the selected prototypes decreases. This makes sense because the most reliable samples have been previously selected. However, the kNN policy generally obtains better results from the first iteration, obtaining on average (for all iterations) an improvement of 6.36 % with respect to the confidence policy. This improvement is significantly greater in the last iterations, obtaining an increase of up to 24.85 % between the result of the confidence policy and the best result obtained with kNN policy.

The role of the parameter k is also illustrated in Fig. 9, where better results are attained as k is increased. It is shown that the impact of this parameter is more noticeable in the last iterations, where a difference of up to 8.91 % is obtained between $k = 3$ and $k = 150$.

Because the kNN selection policy worked better, this policy was used in all following experiments.

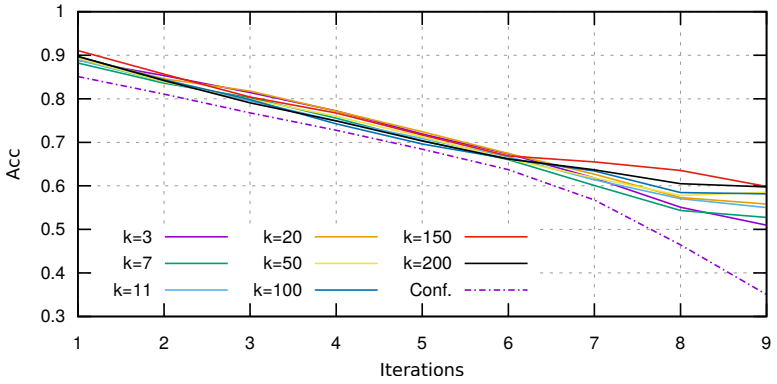


Figure 9: Average accuracy of the labels assigned to the selected prototypes set ($\hat{B}_r$) from the target domain in each iteration according to the selection policy.

5.4. Accuracy on target

In this section, we evaluate the final result obtained through the proposed iDANN method with the best combination of hyper-parameters previously obtained for each of the dataset pairs. In order to check the goodness of the incremental approach, we will compare this result with those obtained by the original DANN method and by applying the incremental approach directly to the CNN backbone used by the iDANN (that is, without applying any DA method). We will denote this last approach as iCNN. Similar to an ablation study, this experiment will allow us to determine whether the greatest contribution to the result obtained should be attributed to a single part of the proposed method (the DA method or the incremental method), or if, on the contrary, it is due to the effectiveness of combining both approaches.

Table 5 reports the results of the experiment, where rows indicate the dataset pairs (source and target) and columns represent the method. Concerning iDANN, we report two results: the accuracy of the labels assigned during the iterative process itself (1), as well as the accuracy using the CNN trained from scratch using only the target samples (once all the target samples have been assigned a label). In addition to DANN, iCNN and iDANN methods, we have also added the results obtained with the neural networks trained just with the source set ('CNN Src.'), as well as the results obtained with the neural networks directly trained with the target set ('CNN Tgt.'). The former serves as baseline, to better assess the impact of the domain-adaptation mechanisms, while the second represents the upper bound of accuracy.

The first thing to remark is that the worst results obtained by the baseline ('CNN Src.') come from the combinations of single-digit datasets (MNIST, MNIST-M) as source and complex digit datasets (SVHN, Syn Numbers) as target. Furthermore, the best results from the baseline are reported for combinations where the source and target are similar (MNIST-M $\rightarrow$ MNIST, SVHN $\rightarrow$ Syn Numbers).

The original DANN method outperforms the results obtained by using the baseline network ('CNN Src.') by 10.7 %, on average, obtaining the most significant improvement for the combinations of Syn Numbers → MNIST (improving by 29.31 %). It is also noticeable the impact of DANN when the dataset pair consists of similar tasks with the most complex one as target, such as MNIST → MNIST-M—improvement of 23 %—or Syn Signs → GTSRB—improvement of 15.49 %. These results for DANN have been obtained using our own implementation, following the details given in the original paper. We observed that the accuracy matches approximately that reported by the authors (for the 4 combinations they considered), and so we assume that our implementation is correct. We can therefore faithfully report the performance in all source-target combinations of our experiments.

As regards the results obtained by the iCNN—that is, applying an incremental self-labeling approach with the underlying CNN)—it is observed an average improvement of 15.1 % with respect to the baseline result ('CNN Src.') and 4.4 % with respect to the classical DANN method. This means that the contribution of the incremental method is slightly higher than that of the DANN, on average; however, if we look at the individual results, the DANN method outperforms the iCNN in some combinations (such as MNIST → MNIST-M or MNIST-M → SVHN). In any case, the iCNN performance is clearly below that offered by the iDANN method.

Concerning the labels assigned during the proposed incremental approach iDANN ($D_T\text{Acc.}^{(1)}$), the first thing to note is its improvement with respect to the underlying DANN method, which is around 16 %, on average. In the best case, this improvement reaches values around 33 %, 35 % and 36 % for the Syn Numbers → MNIST-M, MNIST-M → Syn Numbers, and MNIST → Syn Numbers pairs, respectively. If we compare the result of iDANN with that obtained by iCNN, we see that the improvement is slightly less (12 %). The greatest improvement are obtained in the same cases (30 %, 28 %, and 32 %, respectively). These confirm the goodness of our strategy, because the result obtained by combining the incremental approach and the DANN clearly exceeds that obtained by considering these approaches separately. If we compare the average improvement obtained from DANN and iCNN independently with respect to the baseline (10.7 % and 15.1 %, respectively), we observe that it is below than that obtained by iDANN (26.7 %) by a wide margin. Note that even the arithmetic sum of the independent improvements of iCNN and DANN does not reach that of iDANN.

Finally, if the CNN is trained from scratch with the target labels that have been automatically assigned by the iDANN ($D_T\text{Acc.}^{(2)}$), it can further improve the results up to 1.64 %, on average, and up to 5.5 % in the best case (MNIST-M → Syn Numbers). It should be noted that in some specific combinations, this approach slightly outperforms the CNN trained with the correct target labels (for example, MNIST-M → MNIST or Syn Numbers → SVHN). It might happen that the incorrectly assigned labels of the iDANN process act as a regularizer that alleviates some overfitting.

Table 5: Accuracy (%) over the target dataset for different strategies: 'CNN Src.' indicates a neural network trained only with source samples; 'DANN' denotes the original DANN strategy; 'iCNN' indicates an incremental self-labeling strategy, without domain adaptation; 'iDANN' yields two results from the incremental strategy: D_TAcc.$^{(1)}$ refers to the accuracy of the labels assigned to the target samples during the iterative process, while D_TAcc.$^{(2)}$ refers to the classification after training a new CNN from scratch using the labels assigned to the target samples; and 'CNN Tgt.' denotes a CNN trained using the ground truth of the target samples.

Source	**Target**	**CNN Src.**	**DANN**	**iCNN**	**iDANN**		**CNN Tgt.**
		$\boldsymbol{D_T}$ **Acc.**	$\boldsymbol{D_T}$ **Acc.**	$\boldsymbol{D_T}$ **Acc.**	$\boldsymbol{D_T}$**Acc.**$^{(1)}$	$\boldsymbol{D_T}$**Acc.**$^{(2)}$	$\boldsymbol{D_T}$ **Acc.**
MNIST	MNIST-M	55.71	78.70	71.05	96.09	96.67	97.34
	SVHN	16.26	31.32	31.22	35.83	36.49	90.93
	Syn Numbers	32.14	44.66	48.76	80.79	84.82	99.34
MNIST-M	MNIST	97.95	98.65	98.95	99.04	99.59	98.94
	SVHN	32.91	41.41	39.93	61.87	61.89	90.93
	Syn Numbers	46.34	54.02	61.94	89.49	94.99	99.34
SVHN	MNIST	59.04	67.08	71.26	82.72	84.50	98.94
	MNIST-M	43.49	47.42	62.88	66.40	67.62	97.34
	Syn Numbers	88.42	89.56	95.26	96.43	98.10	99.34
Syn Numbers	MNIST	60.04	89.35	96.74	98.13	99.35	98.94
	MNIST-M	41.84	54.38	57.36	87.10	90.26	97.34
	SVHN	85.16	87.24	89.06	91.42	91.95	90.93
GTSRB	Syn signs	76.39	86.22	97.21	98.28	98.57	99.74
Syn signs	GTSRB	69.79	85.28	95.01	96.31	98.00	97.89
Average		57.53	68.23	72.62	84.28	85.91	96.95

5.5. Comparison with the state of the art

To conclude the results section, we present below a comparison with other domain adaptation strategies from the state of the art for the digits benchmarks. In these works, not all possible combinations of source-target pairs are considered. We show in Table 6 the results reported in the literature[1], along with the results obtained by our proposal (iDANN). A brief description of the competing methods was provided in Section 2. Readers are referred to the corresponding references for further details.

Furthermore, with the intention of making a fair comparison, the proposed method has also been evaluated using ResNet-101 [8] as backbone, and also using the same backbone that was used in the

[1]Unlike the results of the previous section, the DANN values of Table 6 are those reported in the original paper [19].

DTA approach. In the case of ResNet-101, the images have been rescaled to 128×128 px, starting with the pre-trained weights from ImageNet [42].

These results reveal that our method yields the best performance in 4 out of 7 source-target pairs when comparing with the best methods from the state of the art. The performance of iDANN is especially remarkable in the case of MNIST $\rightarrow$ Syn Num, where the improvement reaches around 30 % compared to the literature. For the cases in which our proposal does not attain the best result, we observe a dissimilar performance: it is still very competitive for the MNIST $\rightarrow$ SVHN pair, whereas it is outperformed for the SVHN $\rightarrow$ MNIST pair. When all the results are good, the improvement is relative, but when there is enough margin, the improvement is quite remarkable (as in the case of MNIST $\rightarrow$ Syn Num).

Table 6: Comparison of accuracy (%) between state-of-the-art DA approaches and iDANN over digits datasets. The first two rows denote the source and target dataset, respectively. The best result for each combination (column) is highlighted in bold typeface, while the second best is underlined. Empty cells indicate that the result is not reported in the literature.

	MNIST			**SVHN**		**Syn Num**	
Methods	**MNIST-M**	**SVHN**	**Syn Num**	**MNIST**	**Syn Num**	**MNIST**	**SVHN**
SA [29]	56.9	–	–	59.32	–	–	86.44
DRCN [21]	–	<u>40.05</u>	–	81.97	–	–	–
DSN [23]	83.2	–	–	82.7	–	–	91.2
DANN [19]	76.66	12.4	22.9	73.85	96.9	87.6	91.09
D-CORAL [30]	–	35	55.8	76.3	95.5	89.9	78.8
ADDA [25]	–	–	–	76	–	–	–
ADA [24]	–	12.9	34.8	96.3	95.5	97.1	88.1
VADA [20]	–	18.6	45.9	92.9	96.8	96.2	85.3
DeepJDOT [27]	92.4	–	–	<u>96.7</u>	–	–	–
DTA [26]	–	–	–	**99.4**	–	–	–
Asymmetric [15]	94.2	**52.8**	–	86.2	–	–	**93.1**
CDAN [22]	–	–	–	94.3	–	–	–
iDANN	<u>96.67</u>	36.49	<u>84.82</u>	84.50	**98.10**	**99.35**	91.95
iDANN - DTA	94.25	36.46	79.37	86.14	97.12	98.18	90.06
iDANN - ResNet-101	**96.73**	38.45	**85.02**	85.99	<u>98.01</u>	<u>99.33</u>	<u>92.21</u>

Furthermore, it should be noted that many of the compared methods propose specific CNN architectures for each combination of datasets and/or focus on optimizing the result for a particular combination, such as DTA, DeepJDOT, or CDAN. Inspecting the results, it is observed that no

method is the best in all cases. In our case, we utilized the topologies proposed in the original DANN paper, in addition to two state-of-the-art topologies for providing a fair comparison. It could be assumed that, if we pursue a neural architecture or tune the training hyper-parameters specifically for each of the source-target pairs, our results will surely improve.

Finally, to provide a better evaluation of the competitiveness of our method, we report in Table 7 the same comparison over the Office-31 dataset [43]. This is a widely considered benchmark for visual domain adaptation, with $4,652$ images and 31 categories, collected from three distinct domains: Amazon (A), Webcam (W) and DSLR (D). We evaluate all methods on six transfer tasks: A $\rightarrow$ W, D $\rightarrow$ W, W $\rightarrow$ D, A $\rightarrow$ D, D $\rightarrow$ A, and W $\rightarrow$ A.

In this case, given the higher complexity of the classification task itself, our iDANN method was trained with ResNet-101 as backbone, starting with the pre-trained weights from ImageNet, and applying data augmentation. The transformations applied were randomly selected from the following set: horizontal flips, horizontal and vertical shifts ($[-5, 5]\%$ of the image size), zoom ($[-5, 5]\%$ of the original image size), and rotations (in the range $[-30°, 30°]$).

Similarly to the previous case with digits, iDANN is always among the best results or at least very close. In fact, it gets the second best average (behind CRST) by a narrow margin. These results showcase that our method is competitive in more complex datasets as well, although it was developed in a rather general way.

Table 7: Comparison of accuracy (%) between state-of-the-art DA approaches and iDANN over Office-31 dataset. The best result for each combination (column) is highlighted in bold typeface, while the second best is underlined.

Method	**A→W**	**D→W**	**W→D**	**A→D**	**D→A**	**W→A**	**Avg.**
ResNet-50 [8]	68.4	96.7	99.3	68.9	62.5	60.7	76.1
DAN [44]	80.5	97.1	99.6	78.6	63.6	62.8	80.4
DANN [19]	82.0	96.9	99.1	79.7	68.2	67.4	82.2
ADDA [25]	86.2	96.2	98.4	77.8	69.5	68.9	82.9
JAN [45]	85.4	97.4	<u>99.8</u>	84.7	68.6	70.0	84.3
GTA [46]	<u>89.5</u>	97.9	<u>99.8</u>	87.7	<u>72.8</u>	**71.4**	86.5
CBST [47]	87.8	<u>98.5</u>	**100.0**	86.5	71.2	<u>70.9</u>	85.8
CRST [17]	89.4	**98.9**	**100.0**	<u>88.7</u>	72.6	<u>70.9</u>	**86.8**
CDAN [22]	**93.1**	98.2	**100.0**	**89.8**	70.1	68.0	86.6
iDANN	<u>89.5</u>	<u>98.5</u>	**100.0**	88.6	**73.1**	70.5	<u>86.7</u>

It is, therefore, empirically demonstrated that the iDANN approach offers a very stable behavior, with a competitive performance in all cases considered and performing the best in many of them.

6. Conclusions and Future Work

This paper proposes an incremental strategy to the problem of domain adaptation with deep neural networks. Our approach is built upon an existing domain adaptation approach, combined with a self-labeling heuristic that, in each iteration, decides which prototypes of the target set can be added to the training set by considering the label provided by the neural network. These ideas collaborate during the learning process: by moving the self-labeled samples from the target to the source set, the adaptation algorithm is forced to look for new features after each iteration to discriminate between domains. Two selection policies for the self-labeling step were proposed: one directly based on the confidence given by the network to the prediction and another based on geometric properties of the learned feature space. We observed that the latter reported a better performance, especially in the last iterations of the algorithm. In addition, we consider a final stage in which the labeled target set is used to train a new neural network with label smoothing.

Our experiments were performed on various corpora and using several configurations of the neural network. From the results, we conclude that the incremental approach outperforms the underlying DANN model, as well as other state-of-the-art methods. It is interesting to note that, in some cases, the iDANN approach improves on the result obtained with the CNN trained directly with the ground-truth data of the target set, which could indicate that the incremental process also serves as a regularizer that leads to greater robustness. Furthermore, unlike the classic DANN, our approach improves results when domains are similar and helps keeping the accuracy for the source domain. We also observed a greater training stability and less dependence on the hyper-parameters set.

As future work, a primary objective would be to establish a well-principled stop criterion that allows us to detect when the prediction over the target samples is not reliable. In addition, we want to extend the experiments to other types of input types (such as sequences), as well as to study the behavior of the incremental strategy when the underlying DA method is different—given that there currently exist several alternatives for this challenge. Note that our incremental approach is independent of the underlying DA model considered, and so it could be adopted as a generic strategy that might improve to the same extent as the underlying DA algorithm improves.

Acknowledgment

The first two authors thank the support from the Spanish Ministry HISPAMUS project TIN2017-86576-R and the University of Alicante through project GRE19-04.

References

[1] R. O. Duda, P. E. Hart, D. G. Stork, Pattern classification, 2nd Edition, Wiley, 2001.

[2] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, MIT Press, 2016, http://www.deeplearningbook.org.

[3] L. Shao, F. Zhu, X. Li, Transfer learning for visual categorization: A survey, IEEE Transactions on Neural Networks and Learning Systems 26 (5) (2014) 1019–1034.

[4] M. Wang, W. Deng, Deep visual domain adaptation: A survey, Neurocomputing 312 (2018) 135 – 153.

[5] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, Nature 521 (7553) (2015) 436.

[6] I. Triguero, S. García, F. Herrera, Self-labeled techniques for semi-supervised learning: taxonomy, software and empirical study, Knowledge and Information systems 42 (2) (2015) 245–284.

[7] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, in: 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings, 2015.

[8] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, USA, June, 2016, pp. 770–778.

[9] J. Yosinski, J. Clune, Y. Bengio, H. Lipson, How transferable are features in deep neural networks?, in: Advances in neural information processing systems (NIPS), 2014, pp. 3320–3328.

[10] L. Cheng, S. J. Pan, Semi-supervised domain adaptation on manifolds, IEEE Transactions on Neural Networks and Learning Systems 25 (12) (2014) 2240–2249. doi:10.1109/TNNLS.2014.2308325.

[11] T. Yao, Y. Pan, C.-W. Ngo, H. Li, T. Mei, Semi-supervised domain adaptation with subspace learning for visual recognition, in: Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR), 2015, pp. 2142–2150.

[12] K. Saito, D. Kim, S. Sclaroff, T. Darrell, K. Saenko, Semi-supervised domain adaptation via minimax entropy, in: The IEEE International Conference on Computer Vision (ICCV), 2019.

[13] W. M. Kouw, M. Loog, A review of domain adaptation without target labels, IEEE Transactions on Pattern Analysis and Machine Intelligence (2019) 1–1.

[14] K. Bousmalis, N. Silberman, D. Dohan, D. Erhan, D. Krishnan, Unsupervised pixel-level domain adaptation with generative adversarial networks, in: IEEE Conference on Computer Vision and Pattern Recognition, CVPR, Honolulu, USA, July, 2017, pp. 95–104.

[15] K. Saito, Y. Ushiku, T. Harada, Asymmetric tri-training for unsupervised domain adaptation, in: Proceedings of the 34th International Conference on Machine Learning-Volume 70, JMLR. org, 2017, pp. 2988–2997.

[16] N. Inoue, R. Furuta, T. Yamasaki, K. Aizawa, Cross-domain weakly-supervised object detection through progressive domain adaptation, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2018, pp. 5001–5009.

[17] Y. Zou, Z. Yu, X. Liu, B. Kumar, J. Wang, Confidence regularized self-training, in: Proceedings of the IEEE International Conference on Computer Vision, 2019, pp. 5982–5991.

[18] S. Ben-David, J. Blitzer, K. Crammer, F. Pereira, Analysis of representations for domain adaptation, in: B. Schölkopf, J. C. Platt, T. Hoffman (Eds.), Advances in Neural Information Processing Systems 19 (NIPS), MIT Press, 2007, pp. 137–144.

[19] Y. Ganin, E. Ustinova, H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, M. March, V. Lempitsky, Domain-adversarial training of neural networks, Journal of Machine Learning Research 17 (59) (2016) 1–35.

[20] R. Shu, H. Bui, H. Narui, S. Ermon, A DIRT-t approach to unsupervised domain adaptation, in: International Conference on Learning Representations, 2018.

[21] M. Ghifary, W. B. Kleijn, M. Zhang, D. Balduzzi, W. Li, Deep reconstruction-classification networks for unsupervised domain adaptation, in: Computer Vision – ECCV, Springer International Publishing, Cham, 2016, pp. 597–613.

[22] M. Long, Z. CAO, J. Wang, M. I. Jordan, Conditional adversarial domain adaptation, in: Advances in Neural Information Processing Systems 31 (NIPS), Curran Associates, Inc., 2018, pp. 1640–1650.

[23] K. Bousmalis, G. Trigeorgis, N. Silberman, D. Krishnan, D. Erhan, Domain separation networks, in: Advances in Neural Information Processing Systems (NIPS), Curran Associates, Inc., 2016, pp. 343–351.

[24] P. Haeusser, T. Frerix, A. Mordvintsev, D. Cremers, Associative domain adaptation, in: The IEEE International Conference on Computer Vision (ICCV), 2017.

[25] E. Tzeng, J. Hoffman, K. Saenko, T. Darrell, Adversarial discriminative domain adaptation, in: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017, pp. 2962–2971.

[26] S. Lee, D. Kim, N. Kim, S.-G. Jeong, Drop to adapt: Learning discriminative features for unsupervised domain adaptation, in: The IEEE International Conference on Computer Vision (ICCV), 2019.

[27] B. B. Damodaran, B. Kellenberger, R. Flamary, D. Tuia, N. Courty, Deepjdot: Deep joint distribution optimal transport for unsupervised domain adaptation, in: Computer Vision – ECCV, Springer International Publishing, Cham, 2018, pp. 467–483.

[28] C. Villani, Optimal Transport Old and New, Springer, 2009.

[29] B. Fernando, A. Habrard, M. Sebban, T. Tuytelaars, Unsupervised visual domain adaptation using subspace alignment, in: 2013 IEEE International Conference on Computer Vision (ICCV), 2013, pp. 2960–2967.

[30] B. Sun, K. Saenko, Deep coral: Correlation alignment for deep domain adaptation, in: G. Hua, H. Jégou (Eds.), Computer Vision – ECCV 2016 Workshops, Springer International Publishing, Cham, 2016, pp. 443–450.

[31] S. Ben-David, J. Blitzer, K. Crammer, A. Kulesza, F. Pereira, J. W. Vaughan, A theory of learning from different domains, Machine Learning 79 (1) (2010) 151–175. doi:10.1007/s10994-009-5152-4.

[32] J. S. Bridle, Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition, in: F. F. Soulié, J. Hérault (Eds.), Neurocomputing, Springer Berlin Heidelberg, Berlin, Heidelberg, 1990, pp. 227–236.

[33] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, Z. Wojna, Rethinking the inception architecture for computer vision, in: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 2818–2826. doi:10.1109/CVPR.2016.308.

[34] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, in: Proc. of the IEEE, Vol. 86, 1998, pp. 2278–2324.

[35] P. Arbelaez, M. Maire, C. Fowlkes, J. Malik, Contour detection and hierarchical image segmentation, IEEE Transactions on Pattern Analysis and Machine Intelligence 33(5) (2011) 898–916. doi:10.1109/TPAMI.2010.161.

[36] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, A. Y. Ng, Reading digits in natural images with unsupervised feature learning, in: NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011, 2011.

[37] J. Stallkamp, M. Schlipsing, J. Salmen, C. Igel, Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition, Neural Networks 32 (2012) 323 – 332. doi:https://doi.org/10.1016/j.neunet.2012.02.016.

[38] B. Moiseev, A. Konev, A. Chigorin, A. Konushin, Evaluation of traffic sign recognition methods trained on synthetically generated data, in: Advanced Concepts for Intelligent Vision Systems, Springer International Publishing, Cham, 2013, pp. 576–583.

[39] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Dropout: A simple way to prevent neural networks from overfitting, Journal of Machine Learning Research 15 (2014) 1929–1958.

[40] D. Cireşan, U. Meier, J. Masci, J. Schmidhuber, Multi-column deep neural network for traffic sign classification, Neural Networks 32 (2012) 333 – 338, selected Papers from IJCNN 2011.

[41] L. van der Maaten, G. Hinton, Visualizing data using t-SNE, Journal of Machine Learning Research 9 (2008) 2579–2605.

[42] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, L. Fei-Fei, Imagenet: A large-scale hierarchical image database, in: IEEE conference on computer vision and pattern recognition, IEEE, 2009, pp. 248–255.

[43] K. Saenko, B. Kulis, M. Fritz, T. Darrell, Adapting visual category models to new domains, in: Proceedings of the 11th European Conference on Computer Vision, Heraklion, Crete, Greece, September 5-11, 2010, 2010, pp. 213–226.

[44] M. Long, Y. Cao, J. Wang, M. I. Jordan, Learning transferable features with deep adaptation networks, in: Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37, ICML'15, JMLR.org, 2015, p. 97–105.

[45] M. Long, H. Zhu, J. Wang, M. I. Jordan, Deep transfer learning with joint adaptation networks, in: Proceedings of the Int. Conference on Machine Learning (ICML), Vol. 70 of Proceedings of Machine Learning Research, Sydney, Australia, 2017, pp. 2208–2217.

[46] S. Sankaranarayanan, Y. Balaji, C. D. Castillo, R. Chellappa, Generate to adapt: Aligning domains using generative adversarial networks, IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2018) 8503–8512.

[47] Y. Zou, Z. Yu, B. Vijaya Kumar, J. Wang, Unsupervised domain adaptation for semantic segmentation via class-balanced self-training, in: The European Conference on Computer Vision (ECCV), 2018.